Computer Programming 2
(Classes Encapsulation)

Dr. Dina M. Ibrahim
1st year Students
2014-2015

Lecture #7

28/3/2015

Topics

- Date class Example
- Encapsulation
- Using separate files
- Date class Example (Encapsulated)
- Dynamic Memory Allocation.
 - New Operator
 - Delete Operator
- Example on new & delete.
- MyArray Class

Example: using constructors & destructors

```
1   #include<iostream.h>
2   class Date
3   {
4       private:
5           int day,month,year;
6       public:
7           Date( )
8           {   day=0;
9               month=0;
10              year=0000;          }
11          Date(int d, int m, int y)
12          {   day=d;
13              month=m;
14              year=y;          }
15          ~Date()
16          {   cout<<"\nDestructuring The class Date!!!!!!!";          }
17          void display( )
18          {   cout<<"The day is: "<<day<<"\tThe month is: "<<month
19                  <<"\t The year is: "<<year;          }
20  };
```

Example: using constructors & destructors(Cont.)

```
21  int main( )
22  {
23     Date d1;
24     d1.display( );
25     cout<<"\nThe Constructor uses the default Parameters.\n\n";
26     Date d2(01,01,2000);
27     d2.display( );
28     cout<<"\nThe Constructor does not use the default Parameters.";
29     return 0;
30  }
```

```
The day is: 0   The month is: 0  The year is: 0
The Constructor uses the default Parameters.

The day is: 1   The month is: 1  The year is: 2000
The Constructor does not use the default Parameters.
Destructuring The class Date!!!!!!!
Destructuring The class Date!!!!!!!_
```

Encapsulation

- **Encapsulation:** Separating Class specification from implementation. Hiding the implementation details from the program.

- Separating class declaration, member function definitions, and program that uses the class into separate files is considered **good design.**

Using Separate Files

- Place class declaration in a header file that serves as the class specification file. Name the file ***classname*.h** (for example, **Date.h**)
- Place member function definitions in a class implementation file and main program. Name the file ***classname.cpp*** (for example, **Date.cpp**)This file should **#include** the class specification file. (for example, #include "Date.h")
- All programs that use the class must **#include** the class specification file and be compiled and linked with the class implementation file.

Using Separate Files (Date.h) Header file

```
#ifndef __Date__h__
#define __Date__h__
#include<iostream.h>
class Date
{
    private:
        int day,month,year;
    public:
        Date( )                         //default constructor
        {   day=0;      month=0;        year=0000;      }
        Date(int d, int m, int y)   //overloading constructor
        {   day=d;      month=m;        year=y;         }
        ~Date()                         //destructor
        {   cout<<"\nDestructuring The class Date!!!!!!!";          }
        void display()
        {   cout<<"The day is: "<<day<<"\tThe month is: "<<month
                <<"\t The year is: "<<year;         }
};
#endif
```

Using Separate Files (Date.cpp) Source file

```
#include<iostream.h>
#include "Date.h"    //include the class header file
int main()
{
   Date d1;
   d1.display( );
   cout<<"\nThe Constructor uses the default Parameters.\n\n";
   Date d2(01,01,2000);
   d2.display( );
   cout<<"\nThe Constructor does not use the default Parameters.";
   cout<<"\nWe Separate The Date Class from the source file in a header file called Date.h";
   return 0;
}
```

```
The day is: 0   The month is: 0  The year is: 0
The Constructor uses the default Parameters.

The day is: 1   The month is: 1  The year is: 2000
The Constructor does not use the default Parameters.
We Separate The Date Class from the source file in a header file called Date.h
Destructuring The class Date!!!!!!!
Destructuring The class Date!!!!!!!_
```

Using Separate Files

- For more encapsulation we can use three files:
 - A header file for class specification file. (for example, **Date.h**)
 - A Source file for class implementation file. (for example, **Date.cpp**)
 - A Source file for main program . (for example, **Date_Ex.cpp**)

- Do not forget to add the name of the header file at the beginning of both class implementation and main files (for example, #include “Date.h”)

Dynamic Memory Allocation

- All memory needs were determined before program execution by defining the variables needed.

- But there may be cases where the memory needs of a program can only be determined during runtime. For example, when the memory needed depends on user input.

- On these cases, programs need to dynamically allocate memory, for which the C++ language integrates the operators **new** and **delete.**

new Operator

- Dynamic memory is allocated using operator **new**.
- **new** is followed by a data type and, if a sequence of more than one element is required, the number of these within brackets **[]**.
- It returns a pointer to the beginning of the new block of memory allocated. Its syntax is:

pointer* = *new* *data_type

pointer* = *new* *data_type [number_of_elements]

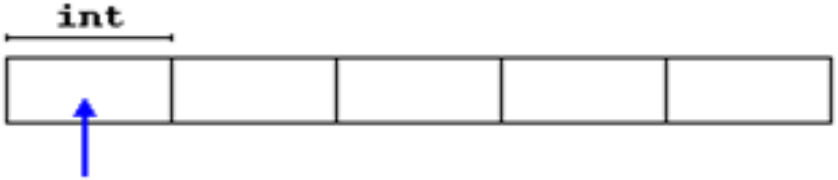
The first expression is used to allocate memory to contain ***single*** element of type ***data_type***. The second one is used to allocate a block (an array) of elements of type ***data_type***, where ***number_of_elements*** is an ***integer*** value representing the amount of these.

new Operator (Cont.)

- For example:

```
int * happy;
happy = new int [5];
```

- In this case, the system dynamically allocates space for five elements of type ***int*** and returns a pointer to the first element of the sequence, which is assigned to ***happy*** (a ***pointer***). Therefore, ***happy*** now points to a valid block of memory with space for five elements of type ***int***.



- Here, ***happy*** is a pointer, and thus, the first element pointed to by happy can be accessed either with the expression ***happy***[0] or the expression ****happy*** (both are equivalent). The second element can be accessed either with ***happy***[1] or *(***happy***+1), and so on...

new Operator (Cont.)

Examples:

- Statement allocates an **integer:**

```
int *pi = new int;
```

- Statement allocates **a character and initializes it:**

```
char *pc = new char('a');
```

- Statement allocates an instance of **Date** using the class's three-argument constructor, and **returns a pointer to it:**

```
Date *pd = new Date(3,12,1990);
```

- Statement allocates **an array of characters:**

```
char *pstr = new char[sizeof(str)];
```

new Operator (Cont.)

- There is a large difference between **declaring a normal array** and **allocating dynamic memory for a block of memory using new**.

- The most important difference is that the size of a regular array needs to be a ***constant expression***, and thus its size has to be determined ***at the moment of designing the program***, before it is run, whereas the dynamic memory allocation performed by ***new allows to assign memory during runtime using any variable value as size.***

- The dynamic memory requested by our program is allocated by the system from the memory **heap**. However, computer memory is a limited resource, and it can be exhausted. Therefore, there are **no guarantees that all requests to allocate memory using operator new are going to be granted by the system.**

new Operator (Exception Handling)

- C++ provides two standard mechanisms to check if the allocation was successful: One is by handling exceptions. Using this method, an exception of type ***bad_alloc*** is thrown when the allocation fails.

- Exceptions are a powerful C++ feature explained later in this course. But for now, you should know that if this exception is thrown and it is not handled by a specific handler, **the program execution is terminated**.

- This exception method is the method used by default by new, and is the one used in a declaration like:

```
happy = new int [5];  // if allocation fails, an exception is thrown  (Error)
```

delete Operator

- In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of operator delete, **whose syntax is:**

```
delete pointer;
delete[ ] pointer;
```

- The first statement releases the memory of a single element allocated using new, and the second one releases the memory allocated for arrays of elements using new and a size in brackets ([]).
- The value passed as argument to delete shall be either a pointer to a memory block previously allocated with new, or a ***null pointer*** (in the case of a *null pointer*, delete produces no effect).

Example: new & delete

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 int main ( )
 5 {
 6   int i, n;
 7   int * p;
 8   cout<< "How many No. would you like to type?
 9   ";
10   cin>>i;
11     p= new int[i];
12   for (n=0; n<i; n++)
13      {
14      cout<< "Enter number: ";
15      cin>> p[n];
16      }
17   cout<< "You have entered: ";
18   for (n=0; n<i; n++)
19      cout<< p[n] << ", ";
20   delete[] p;
21   return 0;
   }
```

```
How many No. would you like to type? 5
Enter number : 10
Enter number : 20
Enter number : 30
Enter number : 40
Enter number : 50
You have entered: 10, 20, 30, 49, 50,
```

Example: Array Class

It is required to design and implement an array class as an ADT, called **MyArray**. The class should provide the following features:

1. ***A constructor that creates a dynamic store for the elements in an array. The default constructor should store up to 100 items of integers.***
2. ***A destructor.***
3. ***A method, Find(), that can check if a given item is in the array or not.***
4. ***A method, Insert(), that can insert a given item into the array.***
5. ***A method, Delete(), that can delete a given element from the array.***
6. ***A method, Display(), that can display the contents of the array.***

Then, design and implement an application program which utilizes the class **MyArray** in the following way. An object of **MyArray** should contains an array of integers with the following values **(77, 99, 44, 55, 22, 88, 11, 00, 66, 33)** by the Insert Function. The program would also ask its user to enter an item from the keyboard that to be deleted from the array. The object should response to the user by display the current elements in the array. **Finally, write a complete C++ program that can accomplish all of the above tasks?**

MyArray Class

```
#ifndef  __MyArray__h__
#define __MyArray__h__
class MyArray
{
 private:
       int *ar;
       int Elements;
       int Search(int Key)
       {  int i;
          for(i=0;i<Elements;i++)
             {  if(ar[i]==Key) return i;
              }
          return -1;
       }
  public:
         MyArray(int size);
         ~MyArray(int size);
         bool Find(int Key);
         void Insert(int Value);
         bool Delete(int Value);
         void Display( );
 };
 #endif
```

MyArray.h

A MyArray Class: The implementation

```
#include "MyArray.h"
  MyArray::MyArray(int size)
    {
       ar=new int[size];
       Elements=0;
     }

  MyArray::~MyArray(int size)
    {
        delete [ ]ar;
     }

  bool MyArray::Find(int Key)
    {
       if (Search(Key)== -1)
             return false; //calling a private method
       return true;
     }
```

A MyArray Class: The implementation(Cont.)

```
void MyArray::Insert(int Value)
  {  ar[Elements]=Value;
     Elements++;  }

bool MyArray::Delete(int Value)
  {  int index, j;
     index=Search(Value);
     if(index != -1)
       {   for(j=index; j<Elements; j++)
              ar[j]=ar[j+1];
           Elements--;
           return true;
       }
      return false;   }

void MyArray::Display( )
   {   int i;
       for(i=0; i<Elements; i++)
           cout<<ar[i]<<" ";
       cout<<endl;
  }
```

MyArray.cpp

Using the MyArray Class

```
#include <iostream>
#include "MyArray.h"
 void main( )
 {
    MyArray ar1(100);
    ar1.Insert(77);  ar1.Insert(99);  ar1.Insert(44);  ar1.Insert(55); ar1.Insert(22);
    ar1.Insert(88);  ar1.Insert(11);  ar1.Insert(00);  ar1.Insert(66); ar1.Insert(33);
    ar1.Display( );
    int Key;
    cout<<"Enter a key for search:";
    cin>>Key;
    if(ar1.Find(Key))
          cout<<"Key "<<Key<<" is found.\n";
    else
          cout <<"Key "<<Key<<" is not found.\n";
    cout<<"Enter a Key to be deleted: "
    cin>>Key;
    ar1.Delete(Key);
    ar1.Display( );
 }
```

Test the program and write the output

Thanks